UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Ada Compiler Validation Summary Report: GEC Software Ltd., VADS Version 5.5, SUN 3/50 Workstation (Host) to GEC 4195 Minicomputer (Target), 880714N1.09135

5. TYPE OF REPORT & PERIOD COVERED
15 July 1989 to 15 July 1990

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
National Computing Centre Limited, Manchester, United Kingdom.

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
National Computing Centre Limited, Manchester, United Kingdom.

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
National Computing Centre Limited, Manchester, United Kingdom.

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)
UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)
Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
GEC Software Ltd., VADS Version 5.5, Wright-Patterson AFB, SUN 3/50 Workstation under SUN UNIX 4.2, Release 3.2 (Host) to GEC 4195 Minicomputer under OS4000 Release 4.17 (Target), ACVC 1.9.

89

5

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

AVF Control Number: AVF-VSR-90502/34

Ada* COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 880714N1.09135
GEC Software Ltd
VADS Version 5.5
SUN 3/50 Workstation x GEC 4195 Minicomputer

Completion of On-site Testing:
15th July 1988

Prepared By:
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
United Kingdom

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

Ada* Compiler Validation Summary Report:

Compiler Name: VADS Version 5.5,

Certificate Number: 880714N1.09135

Host:
SUN 3/50 Workstation under
SUN UNIX 4.2,
Release 3.2

Target:
GEC 4195 Minicomputer under
OS4000 Release 4.17

Testing Completed 15th July 1988 Using ACVC 1.9

This report has been reviewed and is approved.

A E J. Pink

The National Computing Centre Ltd
Jane Pink
Oxford Road
Manchester, M1 7ED
United Kingdom

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

for Edward W Lichtman

Ada Joint Program Office
~~Virginia L. Castor~~ Deputy Director
Director John P. Solomond
Department of Defense
Washington DC 20

* Ada is a registered trademark of the United States Government *(Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-2
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-1
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS 3-4
3.7 ADDITIONAL TESTING INFORMATION 3-5

3.7.1 Prevalidation 3-5
3.7.2 Test Method 3-6
3.7.3 Test Site 3-6

APPENDIX A CONFORMANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada Compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:-

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behaviour is allowed by the Ada Standard.

Testing of this compiler was conducted by NCC under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 15th July 1988 at GEC Software Ltd, 132-135 Long Acre, London WC2E 9AH.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:-

The National Computing Centre Ltd
Oxford Road
Manchester M1 7ED
United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:-

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles sucessfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support are self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated.

A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: VADS Version 5.5,

ACVC Version: 1.9

Certificate Number: 880714N1.09135

Host Computer:

Machine:	SUN 3/50 Workstation
Operating System:	SUN UNIX 4.2 Release 3.2
Memory Size:	4 Mbytes

Target Computer:

Machine:	GEC 4195 Minicomputer
Operating System:	OS4000 Release 4.17
Memory Size:	2 Mbytes

Communications Network: X25/X29 Colour Book Software

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 33 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 10 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, SHORT_FLOAT, and TINY_INTEGER in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

  Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Sometimes NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Sometimes NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round round away from zero (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round to even. (See test C4A014A.)

. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception.(See testC36003A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

. Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behaviour for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 411 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 174 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 139 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	105	1049	1457	14	13	46	2684
Inapplicable	5	2	396	3	5	0	411
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	190	498	540	243	165	98	141	326	137	36	234	3	73	2684
Inapplicable	14	74	134	5	1	0	2	1	0	0	0	0	180	411
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 Tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	C35904A	C37215C	C41402A	CC1311B
	C35904B		C45332A	
E28005C	C35A03E	C37215E	C45614C	BC3105A
C34004A	C35A03R	C37215G	A74016C	AD1A01A
C35502P	C37213H	C37215H	C85018B	CE2401H
A35902C	C37213J	C38102C	C87B04B	CE3208A

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 411 tests were inapplicable for the reasons indicated:

- C35702B uses LONG_FLOAT which is not supported by this implementation.

- A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.

- A39005G uses an array within a record, for which a record representation clause has been stated in the declaration which this compiler does not implicitly pack.

- The following tests use LONG_INTEGER, which is not supported by this compiler:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C		C45631C
C45632C	B52004D	C55B07A	B55B09C	

- C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

- C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.

- D55A03G...H (2 tests) use more than 33 levels of loop nesting which exceeds the capacity of the compiler.

- D64005G uses nested procedures as subunits to a level of 17 which exceeds the capacity of the compiler.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

. C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.

. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

. AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

. The following 174 tests are inapplicable because sequential, text and direct access files are not supported.

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	
CE2204A..B(2	CE2208B	CE2210A	
CE2401A..C(3)	CE2401E..F(2)	CE2404A	
CE2405B	CE2406A	CE2407A	CE2408A
CE2409A	CE2410A	CE2411A	AE3101A
CE3102B	EE3102C	CE3103A	CE3104A
CE3107A	CE3108A..B(2)	CE3109A	CE3110A
CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)	CE3115A
CE3203A		CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

Results of running a subset of these tests showed that the proper exceptions are raised for unsupported file operations.

. The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behaviour. Modifications are made by the AVF in cases where legitimate implementation behaviour prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behaviour that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 28 Class B tests, 109 Class C tests, and 2 Class D tests.

Characterize the tests that needed modification. Where there are several tests with the same characterizations, list them in five columns. The names should be sorted in ascending order, but ignoring the class, i.e., the first character.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B24009A	B24204A	B24204B	B24204C	B25002A
B2A003A	B2A003B	B2A003C	B33301A	B36002A
B37201A	B38003A	B38003B	B38009A	B38009B
B41202A	B44001A	B64001A	B67001A	B67001B
B67001C	B67001D	B91001H	B91003B	B95001A
B97102A	BC1303F	BC3005B		

The following 109 Class C tests and 2 Class D tests were modifiied in order to compensate for the GEC 4195 Minicomputer target's 16K object-module limit. These tests were modified by encapsulating sections of the executable statements in separately compiled procedures which were then called at the place of the statements. Test D64005G was ruled NA as described in section 3.5 as a result of the behavior that was exhibitted after modification; all other tests were successfully executed. The AVO permitted this extensive modification of ACVC 1.9 and ruled the tests passed.

C32001B.ADA C35A04A.ADA C37213E.ADA C46012C.DEP C64018A.ADA

C32001E.ADA	C35A04N.ADA	C37213F.ADA	C46012D.DEP	C67002C.ADA
C32107A.ADA	C35A04Q.ADA	C37213G.ADA	C46012E.DEP	C93003A.ADA
C32112A.ADA	C35A06N.ADA	C37213H.ADA	C46012F.DEP	C94002A.ADA
C32112B.ADA	C35A07N.ADA	C37213J.ADA	C46012G.DEP	C94002G.ADA
C34002A.ADA	C35A07O.ADA	C37215E.ADA	C46012H.DEP	C94008C.ADA
C34003A.ADA	C36104B.ADA	C37215F.ADA	C46012I.DEP	C95008A.ADA
C34007D.ADA	C36205A.ADA	C37215G.ADA	C46012J.DEP	C95067A.ADA
C34007G.ADA	C36205B.ADA	C37215H.ADA	C46012K.DEP	C95084A.ADA
C34007P.ADA	C36205C.ADA	C41103B.ADA	C46044B.ADA	C95085C.ADA
C34007S.ADA	C36205D.ADA	C41203A.ADA	C47007A.ADA	C95087A.ADA
C34007U.ADA	C36205E.ADA	C41203B.ADA	C48008A.ADA	C95087B.ADA
C35502E.ADA	C36205F.ADA	C43103B.ADA	C4A011A.ADA	C95087D.ADA
C35503C.ADA	C36205G.ADA	C45111D.ADA	C52102B.ADA	C95089A.ADA
C35503E.ADA	C36205H.ADA	C45112A.ADA	C52102C.ADA	C96006A.ADA
C35507C.ADA	C36205I.ADA	C45112B.ADA	C52102D.ADA	C9A008A.ADA
C35507E.ADA	C36205J.ADA	C45262A.ADA	C61010A.ADA	CC1221A.ADA
C35507H.ADA	C36205K.ADA	C45282B.ADA	C64005C.ADA	CC1222A.ADA
C35508E.ADA	C37006A.ADA	C45503A.ADA	D64005F0M.ADA	CC1224A.ADA
C355081.ADA	C37007A.ADA	C45503B.DEP	D64005G0M.ADA	CC3601A.ADA
C35A03A.ADA	C37213C.ADA	C46012A.DEP	C64106A.ADA	CC3601C.ADA
C35A03N.ADA	C37213D.ADA	C46012B.DEP	C64106D.ADA	CE3604A.ADA
				CE3704F.ADA

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the VADS Version 5.5 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of the VADS Version 5.5 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a SUN 3/50 Workstation host operating under SUN UNIX 4.2, Release 3.2, and a GEC 4195 Minicomputer target operating under OS4000 Release 4.17. The host and target computers were linked via X25/X29 Colour Book Software.

A magnetic tape containing all tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to themagnetic tape. Tests requiring modifications during the pre- validation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded onto a VAX VMS 11/780 and then transferred via Ethernet onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the SUN 3/50 Workstation, and all executable tests were run on the GEC 4195 Minicomputer. Object files were linked on the host computer, and executable images were transferred to the target computer via X25/X29 Colour Book Software using a SUN 3/160 as a filesaver. Results were printed from the host computer, with results being transferred to the host computer via X25/X29 Colour Book Software using a SUN 3/160 as a filesaver.

The compiler was tested using command scripts provided by GEC Software Ltd and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

Option	Effect
-el	Errors interspersed into the source code

Tests were compiled, linked, and executed (as appropriate) using a 2 host computers and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at GEC Software Ltd, 132-135 Long Acre, London WC2E 9AH and was completed on 15th July 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

GEC Software Limited has submitted the following Declaration of Conformance concerning the VADS Version 5.5.

DECLARATION OF CONFORMANCE

Compiler Implementor: GEC Software Ltd
Ada* Validation Facility: The National Computing Centre Limited, Oxford Rd, Manchester M1 7ED
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name: VADS Version 5.5
Host Architecture ISA: SUN 3/50 Workstation OS&VER #: SUN UNIX 4.2 Release 3.2
Target Architecture ISA: GEC 4195 Minicomputer OS&VER #: OS4000, Release 4.17

Implementor's Declaration

I, the undersigned, representing GEC Software Ltd, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that The Verdix Corporation is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Anne S. Evetts Date: July 22nd 1988

GEC Software limited
Anne Evetts Commercial Manager

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Owner's Declaration

I, the undersigned, representing GEC Software Ltd in conjunction with The Verdix Corporation take full responsibility for the implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Anne S. Evetts Date: 2/4/88

GEC Software Ltd
Anne Evetts Commercial Manager

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the VADS Version 5.5, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

APPENDIX F: Implimentation Dependent Characteristics.

1. Implimentation Dependent Pragmas.

1.1 INLINE_ONLY Pragma

The INLINE_ONLY pragma, when used in the same way as pragma INLINE, indicates to the compiler that the subprogram must always be inlined. This pragma also suppresses the generation of a callable version of the routine which save code space.

1.2 BUILT_IN Pragma

The BUILT_IN pragma is used in the implementation of some predefined Ada packages, but provides no user access. It is used only to implement code bodies for which no actual Ada body can be provided, for example the MACHINE_CODE package.

1.3 SHARE_CODE Pragma

The SHARE_CODE pragma takes the name of a generic instantiation or a generic unit as the first argument and one of the indentifiers TRUE or FALSE as the second argument. This pragma is only allowed immediately at the place of a declarative item in a declarative part or package specification, or after a libary unit in a compilation, but before any subsequent compilation unit.

When the first argument is a generic unit the pragma applies to all instatiations of that generic. When the first argument is the name of a generic instantiation the pragma implies only to the specified instantiation, or overloaded instantiations.

If the second argument is TRUE the compiler will try to share code generated for a generic instantiation with code generated for other instantiations of the same ge[illegible]nen the second argument is FALSE each instatiation will get a unique copy of the generated code. The extent to which code is shared between instatiations depends on this pragma and the kind of generic formal parameters declared for the generic unit.

The name pragma SHARE_BODY is also recognized by the implementation and has the same effect as SHARE_CODE. It is included for compatibility with earlier versions of VADS.

1.4 NO_IMAGE Pragma

The pragma supresses the generation of the image array used for the IMAGE attribute of enumeration types. This eliminates the overhead required to store the array in the executable image.

1.5 EXTERNAL_NAME Pragma

The EXTERNAL_NAME pragma takes the name of the subprogram or variable defined in Ada and allows the user to specify a different external name that may be used to reference the entitity from other languages. The pragma is allowed at the place of a declarative item in a specification and must apply to an object declared earlier in the same package specification.

1.6 INTERFACE_OBJECT Pragma

The INTERFACE_OBJECT pragma takes the name of a variable defined in another language and allows it to be referenced directly in Ada. The pragma will replace all occurrences of the variable name with an external reference to the second, link_argument. The pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification. The object must be declared as a scalar or an access type. The object cannot be any of the following:

a loop variable,
a constant,
an initialized variable,
an array, or
a record.

1.7 IMPLICIT_CODE Pragma

Takes one of the identifiers ON or OFF as the single argument. This pragma is only allowed within a machine code procedure. It specifies that implicit code generated by the compiler to be allowed or disallowed. A warning is issued if OFF is used and any implicit code needs to be generated. The default is ON.

2. Implementation of Predefined Pragmas.

2.1 CONTROLLED

This pragma is recognized by the implementation but has no effect.

2.2 ELABORATE

This pragma is implemented as described in Appendix B of the ADA LRM.

2.3 INLINE

This pragma is implemented as described in Appendix B of the ADA LRM.

2.4 INTERFACE

This pragma supports calls to BABBAGE via the unchecked Interface.

2.5 LIST

This pragma is implemented as described in Appendix B of the ADA LRM.

2.6 MEMORY_SIZE

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

2.7 OPTIMIZE

This pragma is recognized by the implementation but has no effect.

2.8 PACK

This pragma will cause the compiler to choose a non-aligned representation for composite types. It will not cause objects to be packed at the bit level.

2.9 PAGE

This pragma is implemented as described in Appendix B of the ADA LRM.

2.10 PRIORITY

This pragma is implemented as described in Appendix B of the ADA LRM.

2.11 SHARED

This pragma is recognized by the implementation but has no effect.

2.12 STORAGE_UNIT

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means pragmas, the SYSTEM package must be recompiled.

2.13 SUPPRESS

This pragma is implemented as described, except that RANGE_CHECK and DIVISION_CHECK cannot be suppressed.

2.14 SYSTEM_NAME

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

3 Implementation Dependent Attributes

3.1 P'REF

For a prefix that denotes an object, a program unit, a lable, or an entry:

This attribute denotes the effective address of the first of the storage units allocated to P. For a subprogram, package, task unit, or label, it refers to the address of the machine code associated with the corresponding body or statement. For an entry for which and address clause has been given, it refers to the corresponding body or statement. The attribute is of the type OPERAND defined in the package MACHINE_CODE. The attribute is only allowed within a machine code procedure.

See section F.4.8 for more information on the use of this attribute.

(For a package, task unit, or entry, the 'REF attribute is not supported.)

4. Specification of Package SYSTEM

```
package SYSTEM
is
type NAME is ( SUN_CROSS_41XX );
SYSTEM_NAME          : constant NAME := SUN_CROSS_41XX;
STORAGE_UNIT         : constant := 8;
MEMORY_SIZE          : constant := 16_777_216;
-- System-Dependent Named Numbers
MIN_INT              : constant := -2_147_483_648;
MAX_INT              : constant := 2_147_483_647;
MAX_DIGITS                      : constant := 15;
MAX_MANTISSA         : constant := 31;
FINE_DELTA                      : constant := 2.0**(-31);
TICK                 : constant := 0.01;
-- Other System-dependent Declarations
subtype PRIORITY is INTEGER range 0 .. 99;
MAX_REC_SIZE : integer := 10*1024;
type ADDRESS is private;
NO_ADDR : constant ADDRESS;
function PHYSICAL_ADDRESS(I: INTEGER) return ADDRESS;
function ADDR_GT(A, B: ADDRESS) return BOOLEAN;
function ADDR_LT(A, B: ADDRESS) return BOOLEAN;
function ADDR_GE(A, B: ADDRESS) return BOOLEAN;
function ADDR_LE(A, B: ADDRESS) return BOOLEAN;
function ADDR_DIFF(A, B: ADDRESS) return INTEGER;
function INCR_ADDR(A: ADDRESS; INCR: INTEGER) return ADDRESS;
function DECR_ADDR(A: ADDRESS; DECR: INTEGER) return ADDRESS;
function ">"(A, B: ADDRESS) return BOOLEAN renames ADDR_GT;
function "<"(A, B: ADDRESS) return BOOLEAN renames ADDR_LT;
function ">="(A, B: ADDRESS) return BOOLEAN renames ADDR_GE;
function "<="(A, B: ADDRESS) return BOOLEAN renames ADDR_LE;
function "-"(A, B: ADDRESS) return INTEGER renames ADDR_DIFF;
function "+"(A: ADDRESS; INCR: INTEGER) return ADDRESS renames INCR_ADDR;
function "-"(A: ADDRESS; DECR: INTEGER) return ADDRESS renames DECR_ADDR;
pragma inline(ADDR_GT);
pragma inline(ADDR_LT);
pragma inline(ADDR_GE);
pragma inline(ADDR_LE);
pragma inline(ADDR_DIFF);
pragma inline(INCR_ADDR);
pragma inline(DECR_ADDR);
pragma inline(PHYSICAL_ADDRESS);
private
        type ADDRESS is new integer;
        NO_ADDR : constant ADDRESS := 0;
end SYSTEM;
```

5 Restrictions on Reprentation Clauses

5.1 Pragma PACK

Array components less than STORAGE_UNIT bits are packed to the next highest power of 2 bits. Objects and larger components are packed to the nearest whole STORAGE_UNIT. In the absence of pragma PACK record components are padded so as to provide for efficient access by the target hardware, pragma PACK has no other effect. On the storage allocation for record components, a record representation clause is required.

5.2 Length Clauses

Length Clauses are not supported.

5.3 Enumeration Representation Clauses

Enumeration Representation Clauses are supported.

5.4 Record Representation Clauses

For scalar types a representation clause willpack to the number of bits required to represent the range of the subtype. A record representation applied to a composite type will not cause the object to be packed to fit in the space required. An explicit represantation clause must be given for the component type. An error will be issued if there is insufficient space allocated.

5.5 Address Clauses

Address clauses are supported for variables and constants.

5.6 Interrupts

Interrupt entries are not supported.

5.7 Representation Attributes

The ADDRESS attribute is not supported for the following entries:

Packages,
Tasks,
Labels,
Entries.

5.8 Machine Code Insertions

Machine code insertions are supported.

The general definition of the package MACHINE_CODE provides an assembly language interface for the target machine. It provides the necessary record type(s) needed in the code statement, an enumeration type of all the opcode mnemonics, a set of register definitions and a set of addressing mode functions.

The general syntax of a machine code statement is as follows:

```
CODE_n'( opcode, operand );
```

For those opcodes that require no operands, named notation must be used (cf. LRM 4.3(d)).

```
CODE_0'( OP => opcode );
```

The opcode must be an enumeration literal (i.e., it cannot be an object, attribute, or a rename).

An operand can only be an entity defined in the MACHINE_CODE or the 'REF attribute.

The arguments to any of the functions defined in MACHINE_CODE must be static expressions, string literals, or the functions defined in MACHINE_CODE. The 'REF attribute may not be used as an argument in any of these functions.

Inline expansion of machine code procedures is supported.

6. Conventions for Implementation-generated Names

There are no implementation-generated names.

7. Interpretation of Expressions in Address Clauses

Address clauses are supported for constants and variables.

8. Restrictions on Unchecked Conversions

None.

9. Restrictions on Deallocations

None.

10. Implementation Characteristics of I/O Packages

We do not support DIRECT_IO;
We do not support SEQUENTIAL_IO;
We do not support any File IO in TEXT_IO.

11 Implementation Limits

The following limits are actually enforced by implementation. It is not intended to imply that resources up to or even near these limits are available to every program.

11.1 Line Length

The implementation supports a maximum line length of 500 characters including the end of line character.

11.2 Record and Array Sizes

The maximum size of a statically sized array type is 16,000 STORAGE_UNITS. The maximum size of a statically sized record type is 16,000 STORAGE_UNITS. This is the value returned by T'STORAGE_SIZE for a task type T.

11.3 Default Stack for Tasks

In the absence of an explicit STORAGE SIZE length attribute the default collection size for an access type is 16,000 STORAGE_UNITS. This is the value returned by T'STORAGE_SIZE for a task type T.

11.4 Default Collection Size

In the absence of an explicit STORAGE SIZE length attribute the default collection size for an access type is 16,000 STORAGE_UNITS. This is the value returned by T'STORAGE_SIZE for an access type T.

11.5 Limit on Declared Objects

There is an absolute limit of 16,000 STORAGE_UNITS for objects declared statically within a compilation unit. If this value is exceeded the compiler will terminate the compilation with FATAL error message.

11.6 Limit on Compilation Unit Size

There is an absolute limit of 16,384 STORAGE_UNITS for the compilation object size. If this value is exceeded the compiler will terminate with LIMIT error message. 12 Standard specifications

12 IN STANDARD PACKAGE

```
type SHORT_INTEGER      <16 - bit, word integer>;
type INTEGER            <32 - bit, long word integer>;
type SHORT_FLOAT        < 6 - digit, 32 - bit, float>;
type FLOAT              <16 - digit, 64 - bit, float>;
type DURATION           <delta 2.0**(-31) range -2147483.648 .. +2147483.647>;
```

12.1 Attributes of types in STANDARD

SHORT_FLOAT

MACHINE_RADIX	16
MACHINE_MANTISSA	6
MACHINE_EMAX	63
MACHINE_EMIN	-64
MACHINE_ROUNDS	TRUE
MACHINE_OVERFLOWS	TRUE
SIZE	32
FIRST	-1.15792082335569848 63E+77
LAST	7.237005145973115539 6E+75
DIGITS	6
MANTISSA	21
EPSILON	9.536743164062500000 00E-07
EMAX	84
SMALL	2.584939414228211484 0E-26
LARGE	1.934280389046202994 1E+25
SAFE_EMAX	252
SAFE_SMALL	6.908934844075555700 3E-77
SAFE_LARGE	7.237005145973115539 6E+75

FLOAT

MACHINE_RADIX	16
MACHINE_MANTISSA	14
MACHINE_EMAX	63
MACHINE_EMIN	-64
MACHINE_ROUNDS	TRUE
MACHINE_OVERFLOWS	TRUE
SIZE	64
FIRST	-1.157920892373161938 2 E+77
LAST	7.237005577332262013 1 E+75
DIGITS	15
MANTISSA	51
EPSILON	8.881784197001252323 9 E-16
EMAX	204
SMALL	1.944692274331606783 5 E-62
LARGE	2.571100870814383299 1 E+61
SAFE_EMAX	252
SAFE_SMALL	6.908934844077555700 3 E-77
SAFE_LARGE	7.237005577332262013 1 E+75

DURATION

SIZE	32
FIRST	-2147483.648
LAST	2147483.647
DELTA	1.000000000000000E-03
MANTISA	32
SMALL	9.765625000000000E-04
LARGE	4.194303999023430E+06
FORE	8
AFT	3
SAFE_SMALL	9.765625000000000E-04
SAFE_LARGE	4.194303999023430E+06
MACHINE_ROUNDS	TRUE
MACHINE_OVERFLOWS	TRUE

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	A....A1 |----| 498 characters
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	A....A2 |----| 498 characters
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	A....A3A....A |----| |----| 248 250 characters
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	A....A4A....A |----| |----| 248 250 characters
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	0....0298 |----| 496 zeroes
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	0....0690.0 |----| 494 zeroes
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	"A....A" |----| 250 characters

Name_and_Meaning	Value
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	"A....A1" |----| 248 characters
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	479 blanks
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2_147_483_647
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	INVALID
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	.TOOLONGNAME
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	75000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	2147483647.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	INVALID
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	.TOOLONGNAME

Name_and_Meaning	Value
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-75000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	2147483648.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	499
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	2:0....011: |----| 494 zeroes
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	16:0....0F.E: |----| 492 zeroes

Name_and_Meaning	Value
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	"A....A" |----| 497 characters
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	TINY_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ALMP.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target T, raising CONSTRAINT_ERROR.

C35502P: The equality operators in lines 62 and 69 should be inequality operators.

A35902C: Line 17's assignment of the nominal upper bound of a fixed point type to an object of that type raises CONSTRAINT_ERROR for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E: This test assumes that attribute 'MANTISSA' returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C35A03R: This test assumes that attribute 'MANTISSA' returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C: Various discriminant constraints are wrongly expected to be
C37215E: incompatible with type CONS.
C37215G:
C37215H:

C38102C: The fixed-point conversion on line 3 wrongly raises CONSTRAINT_ERROR.

C41402A: 'STORAGE_SIZE' is wrongly applied to an object of an access type.

C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

C45614C: REPORT_IDENT_INT has an argument of the wrong type (LONG_INTEGER).

A74016C: A bound specified in a fixed-point subtype declaration lies
C85018B: outside that calculated for the base type, raising
C87B04B: CONSTRAINT_ERROR. Errors of this sort occur re lines 37 and
CC1311B: 59, 142 and 143, 16 and 48, 252 and 253 of the four tests respectively (and possibly elsewhere).

BC3105A: Lines 159..168 are wrongly expected to be incorrect; they are correct.

AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.

CE2401H: The record aggregates in lines 105 and 117 contain the wrong values.

CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.